Abstract Semantics of Synchronous Languages: The Example ESTEREL[1)]

Manfred Broy
Institut für Informatik
Technische Universität München
D–80290 München, Germany

Abstract

We give an abstract denotational semantic model for the synchronous programming language ESTEREL, which is used for the description and implementation of reactive process control systems. We base this semantics on a functional model of behaviour. We describe the input and output histories of ESTEREL programs by streams carrying sets of signals. We represent the behaviour of an ESTEREL component mathematically by a stream processing function. The main difficulty in giving a semantics to ESTEREL consists in the idea of instantaneous reactions of ESTEREL programs to input signals. In our semantics, we overcome this problem by modelling the causality between the events in every time interval by an individual fixpoint construction. The semantic model fixes the meaning of ESTEREL.

[1)] This work was partially sponsored by the Sonderforschungsbereich 342 "Werkzeuge und Methoden für die Nutzung paralleler Rechnerarchitekturen."

1. Introduction

Synchronous languages and description formalisms have been proposed for the development of process control systems as they are used and needed in reactive embedded systems. There are several proposals of such synchronous languages based on the idea of perfect synchrony. A prominent representative is the language ESTEREL (see [Berry, Gonthier 88], [Berry, Gonthier 92]). Another proposal for the description of reactive systems are statecharts (see [Harel 87]). Both for synchronous languages and for statecharts there is a controversial and to some extent confusing discussion about their semantics.

Synchronous languages are intended for the development of real time process control applications as they can be found in embedded systems. Such process control systems are event driven. Events are represented by signals that are emitted within the environment for instance by sensors or by other parts of the process control system and that are processed by the system. The system generates signals in response that are either internal signals to control the internal information processing activities or external signals issued to the environment for instance to control the actuators. These signals are processed under real time requirements.

To simplify the reasoning about time in synchronous languages the basic idea is that we may abstract from the individual timing and collect a possibly large number of input signals, internal signals, and output signals that occur in one time interval one point in time. We deal with this idea of *perfect synchrony* by looking at the programming language ESTEREL. ESTEREL is an interesting language for programming process control systems in real time applications. It is specifically designed for programming reactive systems covering real time control automata. ESTEREL is deterministic and its designers argue that its features are most adequate for typical process control applications.

Certainly, the conceptional separation of nondeterminism, reactivity and concurrency is a favourable step. One might expect that the semantic theory of a deterministic reactive language such as ESTEREL should become much simpler than that of a nondeterministic one like, for instance, CSP or its practical spin-off OCCAM. However, ESTEREL has also quite involved semantic concepts. It has an informal description of its semantics that is puzzling. Typical phrases are found there such as "*instantaneous reaction*" or an "*action that does not take time*". Such phrases are obviously used to illustrate the level of abstraction that the language ESTEREL offers, but they may make it unnecessarily difficult to explain, understand, and accept the ideas and the semantics of ESTEREL.

We formalise the meaning of ESTEREL by a denotational semantics in the following. The definition of these semantics we see not just an intellectual exercise. We believe that ESTEREL offers an acceptable conceptual programming model only if it turns out that it has a sufficiently simple denotational semantic model that we can use also as a basis for the property-oriented specification of ESTEREL programs and for their verification. Based on the idea of streams, we define a fairly simple denotational model for a slightly simplified version of ESTEREL in the following. In section 2 we introduce the fundamental semantic model, motivate its choice, and give the denotational definition. In section 3 we deal with recursion, feedback, and the existence of fixpoints.

2. Semantic Definition

In this section we define a very abstract denotational semantics of ESTEREL. We do not treat the full language ESTEREL, but only a small, but sufficiently interesting kernel which is close to what is called pure ESTEREL. We do not include variables and assignments into this language kernel. However, we claim that these simply can be treated in our semantic model by adding the denotational concept of states and of environments. The introduction of environments may complicate, however, the fixpoint treatment in the next section. Therefore, we prefer the simple sublanguage to be able to concentrate the more fundamental aspects we are mainly interested in.

2.1 The Idea of Perfect Synchrony

The computational model of ESTEREL may most easily be explained by a model for digital hardware such as the synchronous stream processing model of an interactive component that is pulse driven (see [Fuchs 94]). In this model we use a very fine grained, discrete time scale. In every interval of time on every channel at most one signal can be observed. All components are time guarded in the sense that the output as a reaction to input is delayed at least one time tick with respect to the input event. This leads to a very simple and direct notion of causality between input and output events. This model is much too detailed, however, for a simple understanding of the behaviour of a complex reactive system. Therefore, we go over to the more abstract synchrony model where some of the time differences between input and output are abstracted away.

To explain the behaviour of an ESTEREL component and the abstraction due to the idea of perfect synchrony we distinguish between a *microscopic* and a *macroscopic view*. In the microscopic view we assume such a fine discrete time scale that we can distinguish the difference in the timing of all the signals that are in a causal relationship. In other words, if an event e_1 is causal for an event e_2 then the occurrence time of e_1 is strictly less than the occurrence time of e_2.

In the macroscopic view we use a much coarser time granularity. A set of events is collected at each of the discrete time points. This way certain events that are in a causal relationship and therefore occur in the microscopic view at different time points may be mapped to the same time point in the macroscopic view. With respect to the timing at the microscopic and macroscopic level we speak about the *micro* and the *macro* pulse of the reactive system.

At the micro pulse level as well as on the macro pulse level we model the behaviour of a reactive component in ESTEREL by a stream processing function. The component receives a stream of sets of signals as input and issues a stream of sets of signals as output. The i-th entry in the input stream (or output stream respectively) is a possibly empty set of signals that represents the signals received (or issued respectively) at the i-th time point in the micro level. At the micro level, we do assume neither instantaneous reaction to input nor that an action does not take time. At the micro level we assume that our time granularity is fine enough to observe a reaction to input at time point i only at a later point in time. Thus, at the micro level, every action takes time and reaction comes with some delay. We reflect the causality that governs the interaction of the component by the prefix monotonicity of its behaviour function. This function is a mapping between streams of signals that model the communication between a component and its environment.

Following this model, an input stream consists at the macro level of a stream that carries sequences of sets of signals. Each sequence in the stream represents the sets of signals in the order in which they have been communicated within one interval of the macro pulse. This way we can represent the macro pulse on the input and output streams. This model is used in [Gabreau et al. 93]. It is also useful for the general modelling of switching circuits (see [Fuchs 94]).

In the sequel, we work out a more abstract semantic model for ESTEREL by replacing the *sequences of sets of signals* occurring in the macro level view more abstractly by *sets of signals*. We obtain these streams from the streams considered at the macro level by taking the union of the sets in the sequences of signals at the micro level. This way we obtain a more abstract macro level view from the mentioned macro level view of [Gabreau et al. 93] by an appropriate abstraction. For such an abstraction, we map subsequences that represent the sequences of the sets of messages between the clock signals of the macro level into sets at the macro level. We do this by taking the union over the sets in the sequence.

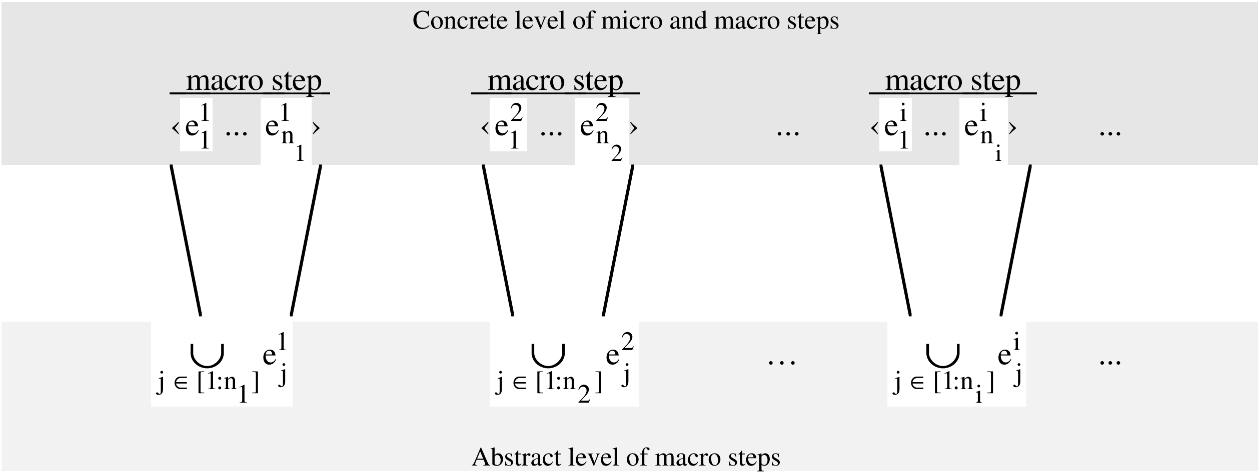

Fig. 1 Concrete Level of Micro and Macro Steps and Abstract Level of Macro Steps

Fig. 1 illustrates the concrete level of micro steps and the abstract level where micro steps are abstracted away.

By this more abstract view at the macro level, an input stream consists of a stream of sets of signals. Each set represents all the signals communicated in one macro step. It seems to be the clue of ESTEREL that such an abstraction is possible (at least for the programs with a proper causality flow) and that we can give a compositional denotational semantics in terms of this more abstract model.

Let us give a mathematical description of what we just explained. For simplicity we assume that each sequence of sets of signals in a macro step is of the same length say $c \in \mathbb{N}$. We could do without a constant length of the sequences of micro steps that form a macro step but this makes the mathematical explanations in the remainder of the section a little easier. At the micro level we work with a stream of sets e_i of signals where for simplicity we assume that every macro cycle contains c micro cycles. We assume that c is chosen sufficiently large to ensure that every computation within a macro cycle comes to a halt. Given a stream

$$e_0 \ \& \ e_1 \ \& \ e_2 \ \& \ e_3 \ \& \ \ldots \ \& \ e_j \ \& \ \ldots$$

at the micro cycle level, we can map this stream onto the following stream on the more abstract macro level:

$$\left(\bigcup_{i=0}^{c*1-1} e_i \right) \& \left(\bigcup_{i=c*1}^{c*2-1} e_i \right) \& \ldots \& \left(\bigcup_{i=c*k}^{c*(k+1)-1} e_i \right) \& \ldots$$

Only throughout the execution of each of the macro cycles, it turns out whether a signal becomes present or definitely remains absent and whether an execution comes to a halt (a stable state) before the time reaches the end of the macro step.

2.2 The Semantic Model

We work with the set of signals M in our semantic model defined as follows:

$$M = E \cup \{†, ‡\}$$

Here E is a predefined set of signals. The specific signal † indicates termination and ‡ indicates pausing. The difference between pausing and termination is essential. Pausing means to stop the activity in a macro step. Termination means that in all the future macro steps there will be no further activity. Some of the signals in E are trap signals. Their role will be explained later. M denotes the set of all ESTEREL signals.

In the translation above we assume that every substream x_k (for $k = 0, 1, ...$) at the micro level that represent a macro step where

$$x_k = S_{c*k} \,\&\, S_{c*k+1} \,\&\, S_{c*k+2} \,\&\, S_{c*k+3} \,\&\, ... \,\&\, S_{c*(k+1)-1}$$

ends with the pause signal ‡ contained in some set S_{c*k+i}. This indicates that the computation has stabilised and halts. Then all successive sets of signals S_{c*k+j} with $j > i$ occurring in the stream x_k till the end of the macro cycle are empty. Otherwise the end of this cycle is not defined and the macro stream is partial and ends abnormally with this macro step. This is indicated by the absence of the pause signal ‡ in the set of events of the macro step.

In our domain, thus we have three classes of streams of sets of signals at the macroscopic level:

- infinite streams of sets of signals all of which contain the signal pause ‡, that indicates that in the respective time interval of the macrocycle the computation stabilised,
- finite streams with well-defined ending that consist of a sequence of sets of signals, where all sets contain the pause signal ‡ except the last one which contains the termination signal †, that indicates that the computation in the last time interval terminated,
- finite streams of sets of signals, where all sets contain the pause signal ‡ except the last one which contains neither the signal ‡ nor the signal †, which indicates that in the last time interval the computation did neither stabilise nor terminate.

The last mentioned streams are called *partial*, since they represent computations that did not end properly. Therefore, for them the signal information is considered as to be incomplete.

We define the set of communication histories E^{∇} over the set of signals E as follows (by $\wp(M)$ we denote the power set over the set M and by M^{ω} we denote the set of streams over the set M, for a short introduction of streams see appendix):

$$E^{\nabla} = \{\, x \in \wp(M)^{\omega}: \quad \#x > 0 \wedge$$
$$(\forall\, j \in \mathbb{N}: 0 < j < \#x \Rightarrow \ddagger \in x.j \wedge \neg(\dagger \in x.j)) \wedge$$
$$(\#x < \infty \Rightarrow \neg(\ddagger \in x.\#x))\}$$

Recall that an ESTEREL history is a nonempty stream of sets of signals that is either infinite and every set contains a pause signal or it is finite and all of its sets contain the pause signal besides its last one. A finite stream therefore ends with a set that does not contain a pause signal. The empty stream is excluded.

The meaning of an ESTEREL component (at the macro level) is represented by a function

$$f: E^{\nabla} \rightarrow E^{\nabla}$$

that maps streams carrying sets of signals onto streams carrying sets of signals. Every set on these streams represents all signals communicated in a complete macro step.

Since the meaning of ESTEREL constructs will be defined with the help of recursion, we have to turn the set

$$E^{\nabla}$$

into a domain to be able to solve fixpoint equations. A domain is a partial ordered set with a least element where every directed set has a least upper bound. Therefore, we need an appropriate partial order that expresses the options to form a fixpoint at the micro level (represented by inclusion order for sets of signals) as well at the macro level (represented by the prefix order on streams). Combining both orderings, we use the partial order Æ_S specified for streams $x, y \in E^{\nabla}$ by the following equation:

$$x \;\text{Æ}_S\; y \;\Leftrightarrow\; x = y \vee$$
$$\forall\, j \in \mathbb{N}\backslash\{0\}: j \leq \#x \Rightarrow (x.j \subseteq y.j \wedge ((\dagger \in x.j \vee \ddagger \in x.j) \Rightarrow x.j = y.j)$$

Informally speaking, a stream x is an approximation of a stream y if both streams are identical or if x is of the finite length j (and y is at least of length j) and all sets in x coincide with the respective sets in y besides the last one that is at least a subset of the j-th set in the stream y, provided this set does not contain the signals ‡ or †. Otherwise, also these sets have to coincide. Since both ‡ and † indicate that the computation in the respective macro step has terminated and therefore the set of signals cannot grow anymore.

The ordering Æ_S is obtained from the classical prefix ordering on streams by the abstraction described above if we assume that we take the union over a number of successive sets of streams that end in each macro cycle with one of the signals † or ‡[1)]. The least element in E^{∇} with respect to

[1)] In fact, the ordering reflects the time flow which requires that the computation in a time interval has to come to a halt (or terminate) before we proceed to the next interval.

the ordering Æ_s is the stream ‹Ø›. Note that the empty stream is not a member of E^{∇}, since every stream is infinite or it has to end with a set that does not contain ‡.

Given streams

$$x = x_1 \mathbin{\&} x_2 \mathbin{\&} \ldots \mathbin{\&} x_n \mathbin{\&} x_{n+1} \mathbin{\&} x_{n+2} \ldots$$

$$y = y_1 \mathbin{\&} y_2 \mathbin{\&} \ldots \mathbin{\&} y_n \mathbin{\&} y_{n+1}$$

and the corresponding abstractions

$$x' = \bigcup_{i=1}^{c} x_i \mathbin{\&} \bigcup_{i=1+c}^{2c} x_i \mathbin{\&} \ldots$$

$$y' = \bigcup_{i=1}^{c} y_i \mathbin{\&} \bigcup_{i=1+c}^{2c} y_i \mathbin{\&} \ldots \mathbin{\&} \bigcup_{i=1+k*c}^{n} y_i$$

where we assume that $1+k*c \le n \le (k+1)*c$ we can compose the orderings on x and y on x' and y'. Obviously, we have y' Æ_s x' provided that the streams y and x are in the prefix order and in all macro cycles (besides the last) all substreams end with † or with ‡.

The relation Æ_s is a partial order. A proof is rather straightforward. This order takes into account both the time flow between the pulse periods (at the macro level) and the accumulation of signals within one pulse (at the micro level).

2.3 Semantic Equations

We give a denotational meaning to ESTEREL programs by associating with every statement a function that maps streams of sets of signals on streams of sets of signals

$$[\![\_]\!]: \langle\text{command}\rangle \to (E^{\nabla} \to E^{\nabla})$$

We extend the well-known operations on sets such as union $\cup$ to streams, and we write

$$\cup^*: E^{\nabla} \times E^{\nabla} \to E^{\nabla}$$

to denote the operation that yields for two streams of sets the stream of sets that we obtain by the union of the elements of the stream. Doing so we take care of the special roles of the signals ‡ and † in the sets. The stream x $\cup^*$ y is defined by the following equations (for simplicity we define $\cup^*$ also for the empty stream; let s, t $\in \wp(E)$, x, y $\in E^{\nabla}$):

$$\langle\rangle \cup^* x = x \cup^* \langle\rangle = x$$

$$(s \mathbin{\&} x) \cup^* (t \mathbin{\&} y) = \begin{array}{lll} \textbf{if} & \dagger \in s & \textbf{then} \quad (s \setminus \{\dagger\} \cup t) \mathbin{\&} y \\ \textbf{elif} & \dagger \in t & \textbf{then} \quad (t \setminus \{\dagger\} \cup s) \mathbin{\&} x \\ \textbf{elif} & \ddagger \in s \wedge \ddagger \in t & \textbf{then} \quad (s \cup t) \mathbin{\&} (x \cup^* y) \\ & & \textbf{else} \quad \langle (s \cup t) \setminus \{\ddagger\} \rangle \\ \textbf{fi} & & \end{array}$$

This definition shows that we join two streams of events by joining their signals E, but halting or termination needs halting or termination of both of them. As already explained, we work with the following specific signals in ESTEREL:

† indicates termination,
‡ indicates pausing.

The difference between pausing and termination is as follows. The signal ‡ indicates that the pause command has been reached within a macro step. The computation comes to a pause and is continued only in the next macro step. Termination is indicated by the signal †. This signal shows that the system has come to a complete stop will not be reactivated in future time intervals.

We define the semantics of the individual statements of ESTEREL inductively on the term structure by the following semantic definitions:

(0) Semantics of the dummy statement `nothing`; it immediately terminates:

$[\![\texttt{nothing}]\!].x = \langle \{\dagger\} \rangle$

(1) Semantics of the pausing statement; it pauses and terminates in the next macro step:

$[\![\texttt{pause}]\!].x = \langle \{\ddagger\} \rangle \hat{}\ \langle \{\dagger\} \rangle$

(2) Semantics of the sequencing of statements:

$[\![\texttt{C}_1 \ ; \ \texttt{C}_2]\!].x = \text{seq}([\![\texttt{C}_1]\!].x, [\![\texttt{C}_2]\!], x)$

where

$\text{seq}(y, f, x) = \langle\rangle$ if $y = \langle\rangle$ and otherwise

$\text{seq}(y, f, x) = \textbf{if}\ \dagger \in \text{ft}.y\ \textbf{then}\ y \cup^* f(x)\ \textbf{else}\ \text{ft}.y \mathbin{\&} \text{seq}(\text{rt}.y, f, \text{rt}.x)\ \textbf{fi}$

(3) Semantics of the parallel statement:

$[\![\texttt{C}_1 \parallel \texttt{C}_2]\!].x = [\![\texttt{C}_1]\!].x \cup^* [\![\texttt{C}_2]\!].x$

(4) Semantics of the infinite loop:

$[\![\texttt{loop C end}]\!].x = \textbf{fix}\ \lambda\, g : \text{seq}([\![\texttt{C}]\!].x, g, x)$

This is only the semantic transliteration of the equation

$[\![$`loop C end`$]\!] = [\![$`C; loop C end`$]\!]$

(5) Semantics of the signal emission: the signal S is emitted and the program terminates immediately:

$[\![$ `emit S` $]\!].x = \langle\{\dagger, S\}\rangle$

(6) Semantics of the test for signal presence:

$[\![$ `present S then` C_1 `else` C_2 `end` $]\!].x =$ **if** $S \in ft.x$ **then** $[\![C_1]\!].x$ **else** $[\![C_2]\!].x$ **fi**

(7) Semantics of the exit from a trap

$[\![$ `exit T` $]\!].x = \langle\{T, \dagger\}\rangle$

(8) Semantics of the trap definition

$[\![$ `trap T in C end` $]\!].x = \text{trapper}([\![C]\!].x, T)$

where

trapper(y, T) = $\langle\rangle$ if y = $\langle\rangle$ and otherwise

trapper(y, T) = **if** $T \in ft.y$ **then** $\langle\{\dagger\} \cup (ft.y \setminus \{T\})\rangle$ **else** ft.y & trapper(rt.y, T) **fi**

(9) Semantics of the suspend

$[\![$ `suspend C when S` $]\!].x =$ **if** $\dagger \in ft.g.x$ **then** g.x

else $\langle ft.g.x\rangle \hat{}\, sus(g \propto x, rt.x)$ **fi**

where

g = $[\![C]\!]$,

and for all functions f and all streams z and y we specify the operator $\propto$ as follows:

$f \propto y = \lambda\, z{:}\ rt.f.(\langle ft.y\rangle \hat{}\, z)$,

sus(f, z) = $\langle\rangle$ if z = $\langle\rangle$ and otherwise

sus(f, z) = **if** $S \in ft.z$ **then** $\langle\{\ddagger\}\rangle \hat{}\, sus(f, rt.z)$

else $\langle ft.f.z\rangle \hat{}\, sus(f \propto z, rt.z)$

fi

It may be more appropriate to make a more explicit distinction between trap signals and other signals, but for our purposes it is sufficient to treat them in this universal way.

2.4 Execution of an ESTEREL Program

An ESTEREL program can run in parallel with another ESTEREL program used to model the environment. The program is executed in a feedback loop with its environment. This feedback makes sure that the signals emitted by the program are also available for it.

We execute an ESTEREL program for an input stream x that comes from the environment as follows. Let

$$f: E^{\nabla} \rightarrow E^{\nabla}$$

be the behaviour of an ESTEREL program. The execution of the program represented by f with input stream x provided by the environment is represented by the stream y where y is defined by the recursive equation

$$y = f(x \cup^{*} y)$$

Of course, we may also model the environment itself by a function

$$g: E^{\nabla} \rightarrow E^{\nabla}$$

and define the behaviour the program in co-operation with the environment by the recursive equation

$$y = f(g(y) \cup^{*} y)$$

In any case, we have to solve a fixpoint equation for the stream y. Since y consists of a sequence of sets and the equation related only these sets we essentially have to solve a fixpoint equation for a set of signals in every time interval of a macro step. The length of the stream y is determined by the fact whether one of these sets that are the result of the interaction within a macro step does not contain the pause signal ‡. In this case, the stream y ends and the least fixpoint is finite. However, a fixpoint only exists if there exist fixpoints for all the sets in the stream y corresponding to the macro steps.

Our semantic works with only one feedback loop due to the fact that we do not include the construct for hiding signals. If we include hiding then we need local feedback loops for feeding back hidden signals.

As a special case we can run an ESTEREL program represented by the function f also in isolation to simulate a closed system. To do this we use the stream

$$x = \{‡\}^{\infty}$$

of pause signals as input from the environment which represents the macro pulse as input and observe the output

$$y = f(x \cup^* y)$$

Since we have for the stream x defined as above $x \cup^* y = y$ for all streams y this is equivalent to the equation

$$y = f.y.$$

This way we can treat system models where the system is modelled by an ESTEREL program Q and the environment is modelled by an ESTEREL program R. Then we model the behaviour by the ESTEREL program

$$E \parallel R$$

which we run in a feedback loop leading to an event stream x where

$$x = [\![E \parallel R]\!].x$$

If the program runs in a loop and a macro cycle contains a pause signal then a new macro cycle is started according to our domain of streams since streams must not end with a set of signals that contains a pause signal ‡.

For completing the semantic definition of our kernel of ESTEREL, we just need to fix the meaning to the fixpoint operator. This is done in the next section.

2.5 Causal Loops and Fixpoints

The semantic definitions make use of the well-known fixpoint operator applied to the functions that are associated as denotations with ESTEREL programs. This is a classical technique in denotational semantics. However, as pointed out by Berry, not all ESTEREL programs show a proper behaviour (are free of *causal* loops). Translated into our setting, this means that such programs are not necessarily monotonic with respect to our ordering.

ESTEREL programs with causal loops do not have a monotonic (w.r.t. $Æ_S$) behaviour function, in general. Consider the following example of a program P with a causal loop:

present S **then** nothing **else** emit S **end**.

We say that this program contains a causal loop, since the signal S on one hand is used as an input for the condition, but on the other hand it is produced as output. According to our semantics, we obtain the meaning of this program by the function $[\![P]\!]$ where $[\![P]\!].x$ is given by the expression

if S $\in$ ft.x **then** {†} **else** {†, S} **fi**

This function is not monotonic in our ordering, since if we increase the set ft.x (by adding the signal S) the result is decreased. It does not have a fixpoint. The assumption of solution of the fixpoint equation leads to a contradiction.

If we analyse our semantic equations we find three types of definitions in the semantic equations:

- the result stream is independent of the input,
- the i-th output set depends on the i-th input set,
- the output stream is obtained by a fixpoint.

This shows that fixpoints are always formed over the sets of signals in one macro time interval[1)]. Therefore, we can concentrate in our treatment of the fixpoint on the sets of signals generated within one time interval. So we only have to study fixpoints for set-valued functions that correspond to the stabilisation of an ESTEREL program in a macro cycle.

In the case of ESTEREL programs that do not have causal loops, a fixpoint can be constructed as follows. If (in every slice of the macro level) there are no cycles in the causality of a function

$$f: E^{\nabla} \to E^{\nabla}$$

this means that for the fixpoint x = f.x in each macro cycle we can find a partial ordering Æ_c (called *causality ordering*) on the set of signals M such that for each pair of signals s_1 and s_2 the proposition

$$s_1 \ \text{Æ}_c \ s_2$$

expresses that the presence or absence of the signal s_1 in the output does not depend on the presence or absence of the signal s_2 in the input[2)] . Since the set of signals is finite we can find a number k and define sets $M_0, \dots, M_k$ for the causality ordering such that[3)]

$$M_i \subseteq M_{i+1}$$

$$s_1 \in M_i \wedge s_2 \in M_{i+1} \backslash M_i \Rightarrow s_1 \ \text{Æ}_c \ s_2$$

$$M = \bigcup \ \{M_i: 0 \le i \le k\}$$

The sets M_i reflect the causality for the signals. All signals in M_0 are independent of all other signals. All signals in $M_{i+1} \backslash M_i$ are independent of all the signals except those in the set M_i. We define the iterated computation of a slice at the macro cycle as follows

[1)] This independence of the computations in the macro cycles would change if we introduced states that allow us to transport results from one macro cycle into the next one.
[2)] Note that the causality ordering on the signal set may be selected differently in each cycle.
[3)] Note the relationship of these sets M_i to the sequences in the micro cycle view.

$X_0 = \emptyset$

$X_{i+1} = f(X_i) \cap M_i$

So the iteration follows the causality. We obtain a chain X_i of sets of signals with $X_i \subseteq X_{i+1}$. Note that for signals s_1, s_2 according to the proper causality flow of the function f with respect to the causality ordering Æ_c we assume for all sets $X \subseteq M$:

$$s_1 \text{ Æ}_c\, s_2 \Rightarrow (s_1 \in f(X)) \equiv (s_1 \in f(X \backslash \{s_2\}))$$

If $s_1 \text{ Æ}_c\, s_2$ then s_2 is not causal for s_1. Therefore the occurrence of the signal s_1 in the set f(X) is independent of the question whether the signal s_2 is a member of f(X) or not. By the assumption of a proper causality flow on the symbols we obtain the following theorem.

Theorem: Let all definitions be as above. Then

$$X = \bigcup_{i \in \mathbb{N}} X_i$$

is the least (with respect to inclusion ordering) fixpoint of f.

Proof: We have to prove that

(1) X is a fixpoint: $X = f(X)$

(2) X is the least fixpoint: $Y = f(Y) \Rightarrow X \subseteq Y$

(1) We obtain the following derivation:

$f(\cup X_i) =$

$\cup (f(\cup X_i) \cap M_i) =$

$\cup (f(X_i) \cap M_i) =$

$\cup X_{i+1} =$

$\cup X_i$

This derivation proves that $X = \cup X_i$ is a fixpoint.

(2) Assume Y is a fixpoint of f. We prove

$X_i \subseteq Y$

for all i by induction on i.

For $i = 0$ we have $X_i = \emptyset$. So trivially $X_0 \subseteq Y$.

Assume now the induction hypothesis $X_i \subseteq Y$. We obtain

X_{i+1}

$= f(X_i) \cap M_i$

$= f(X_i \cap M_i) \cap M_i$

$\subseteq f(Y \cap M_i) \cap M_i$

$= f(Y) \cap M_i$

$= Y \cap M_i$

Since $X_{i+1} \subseteq M_i$ we obtain $X_{i+1} \subseteq Y$. Ÿ

This shows that if there is a proper causality ordering Æ_c for the signals in every macro cycle there exist a least fixpoint that is uniquely determined.

From the papers on ESTEREL it is not made very clear what it means to have a proper causality. We supply such a hopefully intuitively acceptable formal definition. Our semantic definition works for all ESTEREL programs that do not contain causal loops. We just have to compute a fixpoint by an iteration that is obtained by iterations for each macro cycle. According to the theorems above this fixpoint is unique.

3. Conclusions

We have demonstrated how to give a rather abstract denotational semantics to ESTEREL by stream processing functions. We can use this semantic model to write ESTEREL specifications along the lines of the specification and design method FOCUS (see [Broy 91], [FOCUS 92]) FOCUS provides a development method including specification, refinement and verification techniques for functional specification of systems. To carry over this method to ESTEREL on the basis of the introduced semantic model may be treated in a future paper.

The representation of the semantics of ESTEREL programs by stream processing function also allows us to use ESTEREL programs side by side with other system components described by the Focus methodology and also to refine FOCUS specifications into ESTEREL programs. This supports interoperability.

Acknowledgement

It is a pleasure to thank my colleagues Peter Scholz and Jan Philipps for helpful remarks on drafts of this manuscript.

Appendix: Streams

In the following we suppose that interactive systems communicate asynchronously through channels. We use streams to denote histories of communications on channels. Given a set M of messages a stream over M is a finite or infinite sequence of elements from M. By M^* we denote the finite sequences over M. M^* includes the empty sequence which is denoted by $\langle\rangle$.

By M^∞ we denote the infinite sequences over the set M. M^∞ can be understood to be represented by the total mappings from the natural numbers $\mathbb{N}$ into M. We denote the set of streams over the set M by M^ω. Formally we have

$$M^\omega =_{def} M^* \cup M^\infty.$$

We introduce a number of functions on streams that are useful in system descriptions.

A classical operation on streams is the *concatenation* of two streams which we denote by ˆ. The concatenation is a function that takes two streams (say s and t) and produces a stream as result starting with s and continuing with t. Formally the concatenation has the following functionality:

$$.\hat{}. : M^\omega \times M^\omega \to M^\omega.$$

If s is infinite, then concatenating s with t yields s again:

$$s \in M^\infty \Rightarrow s\hat{}t = s.$$

Concatenation is associative and has the empty stream $\langle\rangle$ as its neutral element:

$$r\hat{}(s\hat{}t) = (r\hat{}s)\hat{}t, \qquad \langle\rangle\hat{}s = s = s\hat{}\langle\rangle.$$

For $m \in M$ we denote by $\langle m\rangle$ the one element stream. For keeping our notation simple we extend concatenation also to elements from M (treating them like one element sequences) and to tuples of streams (by concatenating the streams elementwise).

We write for $m \in M$, $s \in M^\omega$ also m & s for $\langle m\rangle\hat{}s$.

On the set M^ω of streams we define a *prefix ordering* Æ. We write s Æ t for streams s and t to express that s is a *prefix* of t. Formally we have for streams s and t:

$$s \text{ Æ } t \quad \text{iff} \quad \exists\, r \in M^{\omega} : s\hat{\ }r = t.$$

The prefix ordering defines a partial ordering on the set M^{ω} of streams. If s Æ t, then we also say that s is an *approximation* of t. The set of streams ordered by Æ is complete in the sense that every directed set $S \subseteq M^{\omega}$ of streams has a *least upper bound* denoted by lub S. A nonempty subset S of a partially ordered set is called *directed*, if

$$\forall\, x, y \in S: \exists\, z \in S: x \text{ Æ } z \wedge y \text{ Æ } z\ .$$

By least upper bounds of directed sets of finite streams we may describe infinite streams. Infinite streams are also of interest as (and can also be described by) fixpoints of prefix monotonic functions. The streams associated with feedback loops in interactive systems correspond to such fixpoints.

A *stream processing function* is a function

$$f: M^{\omega} \rightarrow M^{\omega}$$

that is *prefix monotonic* and *continuous.* The function f is called prefix *monotonic*, if for all streams s and t we have

$$s \text{ Æ } t \Rightarrow f.s \text{ Æ } f.t\ .$$

For better readability we often write for the function application f.x instead of f(x). The function f is called *continuous,* if for all directed sets $S \subseteq M^{\omega}$ of streams we have

$$\text{lub}\ \{f.s: s \in S\} = f.\text{lub}\ S\ .$$

If a function is continuous, then its results for infinite input can be already predicted from its results on all finite approximations of the input.

By $\perp$ we denote the pseudo element which represents the result of diverging computations. We write $M^{\perp}$ for $M \cup \{\perp\}$. Here we assume that $\perp$ is not an element of M. On $M^{\perp}$ we define also a simple partial ordering by:

$$x \text{ Æ } y \quad \text{iff } x = y \vee x = \perp.$$

We use the following functions on streams

$$ft: M^{\omega} \rightarrow M^{\perp},$$

$$rt: M^{\omega} \rightarrow M^{\omega}.$$

They are defined as follows: the function ft selects the first element of a stream, if the stream is not empty. The function rt deletes the first element of a stream, if the stream is not empty. The properties of the functions can be expressed by the following equations that can also be used as defining axioms for them (let $m \in M$, $s \in M^{\omega}$):

$$ft.\langle\rangle = \bot, \qquad rt.\langle\rangle = \langle\rangle, \qquad ft(m\hat{\ }s) = m, \qquad rt(m\hat{\ }s) = s.$$

All the introduced concepts and functions such as the prefix ordering and the concatenation carry over to tuples of streams and functions on streams and tuples of streams by understanding them pointwise.

We denote the function space of (n,m)-ary prefix continuous stream processing functions by:

$$[(M^{\omega})^n \to (M^{\omega})^m].$$

The operations ft and rt are prefix monotonic and continuous, whereas concatenation ˆ as defined above is prefix monotonic and continuous only in its second argument.

The concept of sequences is essential for modelling the stepwise proceeding of computations. When modelling a system component by a state machine or a transition system we obtain finite or infinite computations in the form of sequences of states.

References

[Berry, Gonthier 88]
G. Berry, G. Gonthier: The ESTEREL Synchronous Programming Language: Design, Semantics, Implementation. INRIA, Research Report 842, 1988

[Berry, Gonthier 92]
G. Berry, G. Gonthier: The ESTEREL Synchronous Programming Language: Design, Semantics, Implementation. Science of Computer Programming 19, 1992, 87-152

[Berry 93]
G. Berry: Preemption in Concurrent Systems. In: Proceedings of the FSTTCS '93, LNCS 761, Springer Verlag, 1993, 72-93,

[Berry 95]
G. Berry: The Constructive Semantics of Pure Esterel. Draft Version 1.0, 1995

[Broy 83]
M. Broy: Applicative real time programming. In: Information Processing 83, IFIP World Congress, Paris 1983, North Holland Publ. Company 1983, 259-264

[Broy 85]
M. Broy: Specification and top down design of distributed systems. In: H. Ehrig et al. (eds.): Formal Methods and Software Development. Lecture Notes in Computer Science 186, Springer 1985, 4-28, Revised version in JCSS 34:2/3, 1987, 236-264

[Broy 86]
M. Broy: A theory for nondeterminism, parallelism, communication and concurrency. Habilitation, Fakultät für Mathematik und Informatik der Technischen Universität München, 1982, Revised version in: Theoretical Computer Science 45 (1986) 1-61

[Broy 87a]
M. Broy: Semantics of finite or infinite networks of communicating agents. Distributed Computing 2 (1987), 13-31

[Broy 87b]
M. Broy: Predicative specification for functional programs describing communicating networks. Information Processing Letters 25 (1987) 93-101

[Broy 91]
M. Broy: Functional Specification of Time Sensitive Communicating Systems. REX Workshop.

In: J. W. de Bakker, W.-P. de Roever, G. Rozenberg (eds): Stepwise Refinement of Distributed Systems. Lecture Notes in Computer Science 430, 153-179. Erweiterte Fassung in: ACM Transactions on Software Engineering and Methodology 2:1, Januar 1993, 1-46

[FOCUS 92]
M. Broy, F. Dederichs, C. Dendorfer, M. Fuchs, T.F. Gritzner, R. Weber: The Design of Distributed Systems - an Introduction to FOCUS. Technische Universität München, Institut für Informatik, TUM-I9203, Januar 1992

[Fuchs 94]
M. Fuchs: Technologieabhängigkeit von Spezifikationen digitaler Hardware. Ph.D. Thesis, Technischen Universität München, Fakultät für Informatik. SFB-Report 342/14/94, July 1994

[Gabreau et al. 93]
O. Gabreau, B. Jungen, M. Fuchs: Stream Semantics of ESTEREL. Unpublished Manuscript

[Harel 87]
D. Harel: Statecharts: A Visual Formalism for Complex Systems. Science of Computer Programming 8, 1987, 231 - 274

[Kahn, MacQueen 77]
G. Kahn, D. MacQueen: Coroutines and networks of processes, Proc. IFIP World Congress 1977, 993-998

[Kok 87]
J.N. Kok: A fully abstract semantics for data flow networks. Proc. PARLE, Lecture Notes in Computer Science 259, Berlin-Heidelberg-New York: Springer 1987, 214-219

[Park 80]
D. Park: On the semantics of fair parallelism. In: D. Björner (ed.): Abstract Software Specification. Lecture Notes in Computer Science 86, Berlin-Heidelberg-New York: Springer 1980, 504-526

[Poigne, Holenderski 95]
A. Poigne, L. Holenderski: Boolean Automata for Implementing Pure Esterel. Arbeitspapiere der GMD 964, GMD - Forschungszentrum Informationstechnik GmbH, December 1965